Boolean Exponent Splitting

Michael Tunstall[1] [a], Louiza Papachristodoulou[2] and Kostas Papagiannopoulos[3] [b]

[1]Rambus, 4453 North First Street, Suite 100, San Jose, California, U.S.A.

[2]Fontys University of Applied Sciences, Rachelsmolen 1, Eindhoven, The Netherlands

[3]University of Amsterdam, SNE-CCI, Science Park 904, Amsterdam, The Netherlands

Keywords: Exponent Splitting, Side-channel Attacks, Countermeasures.

Abstract: A typical countermeasure against side-channel attacks consists of masking intermediate values with a random number. In symmetric cryptographic algorithms, Boolean shares of the secret are typically used, whereas in asymmetric algorithms the secret exponent is typically masked using algebraic properties. This paper presents a new exponent splitting technique with minimal impact on performance based on Boolean shares, typically requiring only an extra register and a few register copies per bit. We perform a security evaluation of our algorithms using a mutual information framework and provide proofs that they are secure against first-order side-channel attacks. The side-channel resistance of the proposed algorithms are also practically verified with test vector leakage assessment performed on Xilinx's Zynq zc702 evaluation board.

1 INTRODUCTION

Side-channel analysis as a method of extracting cryptographic keys was first presented by Kocher (Kocher, 1996), who noted that timing differences in the execution time of a modular exponentiation could be used to break instances of RSA (Rivest et al., 1978). Subsequently, Kocher et al. (Kocher et al., 1999) observed that the instantaneous power consumption could reveal information on intermediate states of any cryptographic algorithm, since the instantaneous power consumption has, in many cases, been shown to be proportional to the Hamming weight of the data being manipulated (Brier et al., 2004), and it was later shown that the electromagnetic emanations around a device can be exploited in the same way (Gandolfi et al., 2001; Quisquater and Samyde, 2001).

In public key cryptography, one typically uses countermeasures based on redundant representations to prevent side-channel leakage (Coron, 1999; Win et al., 1998) (referred to as blinding). To protect an exponent used in a group exponentiation one would typically add a random multiple of the order of the group to the exponent, providing a random bitwise representation of the exponent. These countermeasures can provide a strong resistance to Differential Power Analysis (DPA), but are not convenient in some instances. As noted by Smart et al. (Smart et al., 2008), the random value used to blind an exponent needs to have a bit length larger than the longest run of zeros or ones in the bitwise representation of the the order of the group. If we consider ECDSA (National Institute of Standards and Technology (NIST), 2009), for example, the bitwise representations of the orders of the groups used contain long runs of ones making this countermeasure costly.

In this paper, we present a *new countermeasure for exponent splitting*. We describe a method of splitting an exponent into *two Boolean shares*, analogous to the countermeasures that one would use for an implementation of a block cipher and similar to the countermeasures used to prevent address-bit side-channel attacks (Messerges et al., 1999; Messerges and Dabbish, 1999; Itoh et al., 2002). Having embedded devices as our targeted implementation, and an adversary able to get useful information from the length of the exponent or the intermediate values, we provide a number of secure algorithms against a broad range of side-channel attacks.

At the same time, the modifications that are required to a group exponentiation algorithm have negligible effect on the time required to compute the actual group exponentiation, which is a significant advantage over previous examples of exponent splitting (Clavier and Joye, 2001; Ciet and Joye, 2003). In

[a] https://orcid.org/0000-0002-7107-8644

[b] https://orcid.org/0000-0002-5008-1756

Tunstall, M., Papachristodoulou, L. and Papagiannopoulos, K.
Boolean Exponent Splitting.
DOI: 10.5220/0010570903210332
In *Proceedings of the 18th International Conference on Security and Cryptography (SECRYPT 2021)*, pages 321-332
ISBN: 978-989-758-524-1

addition, our method can be efficiently combined with blinding techniques applied to the input to a group exponentiation algorithm, in order to prevent leakage of the intermediate values.

We present an evaluation of the method of Boolean exponent splitting using the information-theoretic framework of Standaert et al. (Standaert et al., 2009) and a Test Vector Leakage Assessment (TVLA) by Goodwill et al. (Goodwill et al., 2011). We investigate the usual leakage models based on data or location leakage and show that an adversary would need either a second-order data attack or a third-order location attack to successfully break the security of our algorithms. In addition, we present for the first time a hybrid model, where data leakage is combined with location leakage, offering new exploitation opportunities. The rich interactions between data and location leakage corroborates the need for holistic countermeasures that encompass a wide spectrum of side-channel attacks.

2 EXPONENT SPLITTING METHODS

The critical operation in public key cryptographic algorithms is exponentiation in a certain group $\mathbb{G}$ of order μ, where the input message $x \in \mathbb{G}$ is raised by a secret exponent κ and the result $y = x^{\kappa}$ is the public output of the algorithm. When implementing a group exponentiation algorithm the exponent is typically blinded by adding some random multiple of the order of the group to the exponent. Trivially, $(r\mu) + \kappa \equiv \kappa \pmod{\mu}$ for $r, \kappa \in \mathbb{Z}$ where r is random. Hence, computing $x^{\kappa + r\mu}$ is equivalent to computing x^{κ}. While this randomizes the bitwise representation of an exponent, the entire exponent is still equivalent to the exponent in a given group. Examples of attacks that have been proposed include analyzing a single trace (from SPA (Kocher et al., 1999) to collisions in manipulated values (Witteman et al., 2011; Kim et al., 2010; Hanley et al., 2015)) or attempting to find collisions in the random values used to then derive a (blinded) exponent (Schindler and Itoh, 2011).

One method that can hinder these attacks, is to split an exponent into two values whose bitwise representations are random. Then one would compute a group exponentiation where the combined effect of the two values is equivalent to that of the desired exponent. Randomly splitting the value that manipulates secret data was proposed initially by Chari et. al in (Chari et al., 1999) as a generic technique to provide provable resistant implementation to side-channel attacks. By randomly splitting every bit of the original computation into m shares, where each share is equiprobably distributed and every proper subset of $m-1$ shares is statistically independent of the encoded bit, the cryptographic computation can then be performed securely by computing only the shares, without ever reconstructing the original bit. The leakage from every computation does not reveal any useful information to the adversary, who needs to perform m attacks to reconstruct the secret. There are several methods of exponent splitting proposed by Clavier and Joye (Clavier and Joye, 2001):

- **Additive Splitting.** For a random integer r with bit-length smaller or equal to the exponent κ, we can define $\kappa = r + (\kappa - r)$. The output of the modular exponentiation $y = x^{\kappa}$ in $\mathbb{G}$ can be computed by $y = x^{r} \cdot x^{\kappa - r}$ in $\mathbb{G}$.
- **Multiplicative Splitting.** For some group $\mathbb{G}$ we can define $k' = k\,r^{-1} \bmod |\mathbb{G}|$ for some integer r. Then the exponentiation $y = x^{k}$ in $\mathbb{G}$ can be computed by using $y = (x^{r})^{k'} \bmod |\mathbb{G}|$.

The same techniques can be applied to scalar multiplication algorithms for elliptic curves (ECs), in order to hide the secret scalar. The problem with these methods of exponent splitting is that one is required to know the order of the group $\mathbb{G}$, which may not be available in some instances. They will also typically double the time required to compute a group exponentiation, because r is required to have a bit-length similar to the exponent. A practical attack by Feix et al. (Feix et al., 2014) demonstrates that a blinded scalar can be determined if r is too small.

A further method described by Ciet and Joye (Ciet and Joye, 2003) is:

- **Euclidean Splitting.** By writing the exponent as $k = \lfloor k/r \rfloor r + k \bmod r$ and letting $s = x^{r}$ for some r, then $y = x^{k}$ can be computed by $y = s^{k'} \times x^{k \bmod r} = (x^{r})^{k'} \times x^{k \bmod r}$, where $k' = \lfloor k/r \rfloor$.

The impact on the time required to compute an exponentiation is lower than the other splitting methods listed above. In fact, in (Ciet and Joye, 2003) the authors evaluated this variant applied to Shamir's double ladder to have the same cost as the 'double-and-add-always' algorithm (equivalent to the 'square-and-multiply-always' for exponentiation). Precomputation of powers of s can reduce the exponentiation cost compared to additive or multiplicative splitting. However, this method has the same constraints as adding a multiple of the group order. That is, r needs to have a bit length larger than the longest run of ones and zeros in k and may have a significant impact on performance (Smart et al., 2008). A secure division algorithm is also required, see, for example, Joye and Villegas (Joye and Villegas, 2002).

3 BOOLEAN EXPONENT SPLITTING METHODS

In this section, we propose methods of exponent splitting based on XOR operation, and how an XOR-split exponent can be applied to the Montgomery powering ladder.

3.1 Montgomery Powering Ladder

The Montgomery Powering Ladder (MPL) was originally proposed as a means of speeding up scalar multiplication over ECs and later shown to be applicable to multiplicative written Abelian groups (Montgomery, 1987; Joye and Yen, 2002). We recall the description of the MPL given by Joye and Yen (Joye and Yen, 2002): We consider the problem of computing $y = x^{\kappa}$ in $\mathbb{G}$ for inputs x and κ. Let $\sum_{i=0}^{n-1} k_i 2^i$ be the binary expansion of κ with bit length n (for ease of expression we shall also denote this as $(k_{n-1},\ldots,k_0)_2$ where convenient). Then, defining $L_j = \sum_{i=j}^{n-1} k_i 2^{i-j}$ and $H_j = L_j + 1$, we have $L_j = 2L_{j+1} + k_j = L_{j+1} + H_{j+1} + k_j - 1 = 2H_{j+1} + k_j - 2$ and so we obtain

$$(L_j, H_j) = \begin{cases} (2L_{j+1}, L_{j+1}+H_{j+1}) & \text{if } k_j = 0, \\ (L_{j+1}+H_{j+1}, 2H_{j+1}) & \text{if } k_j = 1. \end{cases} \tag{1}$$

If we consider one register containing x^{L_j} and another containing x^{H_j} then (1) implies that

$$(x^{L_j}, x^{H_j}) = \begin{cases} \left(\left(x^{L_{j+1}}\right)^2, x^{L_{j+1}} \cdot x^{H_{j+1}}\right) & \text{if } k_j = 0, \\ \left(x^{L_{j+1}} \cdot x^{H_{j+1}}, \left(x^{H_{j+1}}\right)^2\right) & \text{if } k_j = 1. \end{cases} \tag{2}$$

Given that $L_0 = k$ one can build an exponentiation algorithm that requires two group operations per bit of the exponent. Joye and Yen give several different versions, one of which is shown in Algorithm 1. All these methods are highly regular, meaning that a deterministic sequence of operations is executed for an exponent of a given bit length.

Algorithm 1: Montgomery Ladder.

Input: $x \in \mathbb{G}$, an n-bit integer $\kappa = \sum_{i=0}^{n-1} k_i 2^i$
Output: x^{κ}

$R_0 \leftarrow 1_{\mathbb{G}}$; $R_1 \leftarrow x$;
for $i = n-1$ **down to** 0 **do**
  $R_{\neg k_i} \leftarrow R_{k_i} \cdot R_{\neg k_i}$;
  $R_{k_i} \leftarrow (R_{k_i})^2$;
end
return R_0

In applying an XOR-split exponent to MPL we use one share to dictate the address accessed and the other to act as the exponent. That is, we consider (1), where the previous round may provide either (L_j, H_j) or (H_j, L_j) and the computation changed accordingly.

Let $S_{0,j} = L_j$ and $S_{1,j} = H_j$ and $\sum_{i=0}^{n-1} a_i 2^i$ be the binary expansion of A with bit length n (i.e. the same bit length as the exponent). Then we can use the values of a_i to dictate whether a pair of registers holds (L_j, H_j) or (H_j, L_j). Specifically, (1) can be rewritten as:

$$(S_{a_j,j}, S_{\neg a_j,j}) = \begin{cases} (2S_{a_j,j+1}, S_{a_j,j+1} + S_{\neg a_j,j+1}) & \text{if } k_j = 0, \\ (S_{a_j,j+1} + S_{\neg a_j,j+1}, 2S_{\neg a_j,j+1}) & \text{if } k_j = 1. \end{cases} \tag{3}$$

In (3), the values of L_j and H_j are assigned to S in an order dictated by the binary expansion of A. Generating A as a random sequence of bits could provide some side-channel resistance, but does not protect the exponent.

We further consider $\sum_{i=0}^{n-1} a_i 2^i$ and $\sum_{i=0}^{n-1} b_i 2^i$ be the binary expansion of A and B, respectively, where $\kappa = A \oplus B$ of bit length n. We note that, as above, $\sum_{i=0}^{n-1} k_i 2^i$ is the binary expansion of κ and $k_i = a_i \oplus b_i$ for $0 \leq i < n$. Then (3) can be rewritten as:

$$(S_{a_j,j}, S_{\neg a_j,j}) = \begin{cases} (2S_{b_j,j+1}, S_{b_j,j+1} + S_{\neg b_j,j+1}) & \text{if } k_j = 0, \\ (S_{b_j,j+1} + S_{\neg b_j,j+1}, 2S_{\neg b_j,j+1}) & \text{if } k_j = 1. \end{cases} \tag{4}$$

Rather than using the same value to control which order L_j and H_j are assigned and read, we use the bits of A to determine the order L_j and H_j are assigned, and the bits of B to determine the order they are read. The combined effect is that the order L_j and H_j are assigned and read is dictated by the bits of κ.

$$(x^{S_{a_j,j}}, x^{S_{\neg a_j,j}}) = \begin{cases} \left(\left(x^{S_{b_j,j+1}}\right)^2, x^{S_{b_j,j+1}} \cdot x^{S_{\neg b_j,j+1}}\right) \\ \qquad\qquad \text{if } k_j = 0, \\ \left(x^{S_{b_j,j+1}} \cdot x^{S_{\neg b_j,j+1}}, \left(x^{S_{\neg b_j,j+1}}\right)^2\right) \\ \qquad\qquad \text{if } k_j = 1. \end{cases} \tag{5}$$

From which we can define Algorithm 2, which operates in much the same way as the MPL, as it produces a regular sequence of multiplications and squaring operations. However, one more register is required to allow the assignment in line 5 to affect R_0 or R_1. This algorithm is the basis that we use to present the essence of Boolean-split exponent. Algorithm 2 is largely equivalent to an algorithm proposed by Izumi et al. (Izumi et al., 2010) where we set the multiplication in line 4 to operate in a random order as it provides a better resistance to collision attacks, as demonstrated by Kim et al. (Kim et al., 2010). We discuss this further in Section 3.2.

The intermediate states of the registers are not randomized in Algorithm 2 and would require additional countermeasures to provide a secure implementation.

Algorithm 2: Montgomery Ladder with XOR-Split Exponent I.

Input: $x \in \mathbb{G}$, n-bit integers $A = \sum_{i=0}^{n-1} a_i 2^i$ and $B = \sum_{i=0}^{n-1} b_i 2^i$
Output: x^κ where $\kappa = A \oplus B$

$R_0 \leftarrow 1_{\mathbb{G}}$; $R_1 \leftarrow 1_{\mathbb{G}}$; $R_2 \leftarrow 1_{\mathbb{G}}$;
$b' \xleftarrow{R} \{0,1\}$; $R_{\neg b'} \leftarrow x$;
for $i = n-1$ **down to** 0 **do**
  $R_2 \leftarrow R_{a_i} \cdot R_{\neg a_i}$;
  $R_{a_i} \leftarrow \left(R_{(b_i \oplus b') \oplus a_i}\right)^2$;
  $R_{\neg a_i} \leftarrow R_2$;
  $b' \leftarrow b_i$;
end
return $R_{b'}$

For example, inexpensive solutions such as randomizing projective points (Win et al., 1998) or Ebeid and Lambert's blinding method for RSA (Ebeid and Lambert, 2010) can be used (see Section 5). If we assume that the values held in registers $\{R_0, R_1, R_2\}$ do not leak (i.e., we only consider whether the exponent leaks) we can state the following:

Lemma 1. *Assuming that the values held in registers $\{R_0, R_1, R_2\}$ do not leak, an implementation of Algorithm 2 is resistant to first-order side-channel analysis.*

Proof. It suffices to consider each intermediate state and verify that at least one random mask is applied. Verifying this for an entire group exponentiation would be tedious, but can be simplified if we consider two rounds of Algorithm 2. That is, if we consider round m, where $0 \le m \le n-2$, then the following operations are performed:

1. $R_2 \leftarrow R_{a_m} \cdot R_{\neg a_m}$	6. $R_2 \leftarrow R_{a_{m+1}} \cdot R_{\neg a_{m+1}}$
2. $\alpha \leftarrow b_m \oplus b'$	7. $\alpha \leftarrow b_{m+1} \oplus b_m$
3. $\beta \leftarrow \alpha \oplus a_m$	8. $\beta \leftarrow \alpha \oplus a_{m+1}$
4. $R_{a_m} \leftarrow {R_\beta}^2$	9. $R_{a_{m+1}} \leftarrow {R_\beta}^2$
5. $R_{\neg a_m} \leftarrow R_2$	10. $R_{\neg a_{m+1}} \leftarrow R_2$

Let the proposition $\mathcal{P}(n)$ be that round $n > 0$ is resistant to first-order side-channel analysis for the n-th treated bit of the exponent. If we consider the first round, we wish to show $\mathcal{P}(1)$ is true and, in the above code fragment, b' is set to a random value from $\{0,1\}$. Then, it is easy to see that:

- the results of the operations in lines 1, 4, 5, 6, 9 and 10 are dependent on the random values $\{R_0, R_1, R_2\}$.
- the results of the operations in lines 2, 3, 7 and 8 are uniformly distributed on $\{0,1\}$.

If we assume that $\mathcal{P}(m)$ is true for all $m \in \{1, \ldots, n\}$, then we consider $\mathcal{P}(n+1)$ where b' is set to b_n. As b_n is one share of a previously treated exponent bit, it is indistinguishable from a random value from $\{0,1\}$. The above statements regarding the results of the operations apply. Hence, by induction we have shown $\mathcal{P}(n)$ is true for all $n > 0$. To complete the proof, we simply note that only half of the code fragment above will need to be considered in the last round. □

Remark. In (Itoh et al., 2003), the authors present the randomized addressing method (RA), in order to provide protection against address-based DPA and eliminate the correlation between an exponent bit and the register where the result of an operation is stored. In this work, we do not limit our countermeasure to work only against address-based DPA. Our goal is to perform operations on different exponent shares, in a way that an adversary would need a combination of leakages (such as higher-order DPA combined with template attacks) in order to recover the exponent.

3.2 Using Inverses

In this section we propose an algorithm more suited to groups where inversions can be readily computed. Le Duc et al. (Le et al., 2015) propose a straightforward variant of the Montgomery powering ladder that requires the computation of inverses. They note that (1) can be rewritten as:

$$(L_j, H_j) = \begin{cases} (H_j - 1, L_{j+1} + H_{j+1}) & \text{if } k_j = 0, \\ (L_{j+1} + H_{j+1}, L_j + 1) & \text{if } k_j = 1. \end{cases} \tag{6}$$

From which we can define Algorithm 3. If we let $T_{0,j} = L_j$ and $T_{1,j} = H_j$, or $T_{0,j} = H_j$ and $T_{1,j} = L_j$ and store the ordering in another variable we can rewrite (6) as:

$$(T_{0,j}, T_{1,j}) = \begin{cases} (L_j, H_j) & \text{if } k_j = 0 \\ (H_j, L_j) & \text{if } k_j = 1 \end{cases}$$
$$= \begin{cases} (L_{j+1} + H_{j+1}, L_j - 1) & \text{if } k_j = 0, \\ (L_{j+1} + H_{j+1}, L_j + 1) & \text{if } k_j = 1. \end{cases} \tag{7}$$

From which we can define Algorithm 4.
Following the previous notation, we notice that $T_{0,j}$ should contain the sum of the registers in the previous round[1]. Therefore, (7) can be rewritten as follows:

$$(T_{0,j}, T_{1,j}) = \begin{cases} (T_{b',j+1} + T_{\neg b',j+1}, T_{0,j} - 1) & \text{if } k_j = b' = 0, \\ (T_{b',j+1} + T_{\neg b',j+1}, T_{0,j} + 1) & \text{if } k_j = b' = 1. \end{cases} \tag{8}$$

[1]The algorithms are left-to-right, so $j+1$ indicates the round preceding j.

Algorithm 3: Variant with Inverses I.

Input: $x \in \mathbb{G}$, an n-bit integer $\kappa = \sum_{i=0}^{n-1} k_i 2^i$
Output: x^κ

$R_0 \leftarrow 1_{\mathbb{G}}$; $R_1 \leftarrow x$;
$U_0 \leftarrow x^{-1}$; $U_1 \leftarrow x$;
for $i = n-1$ **down to** 0 **do**
  $R_{\neg k_i} \leftarrow R_{k_i} \cdot R_{\neg k_i}$;
  $R_{k_i} \leftarrow R_{\neg k_i} \cdot U_{k_i}$;
end
return R_0

Algorithm 4: Variant with Inverses II.

Input: $x \in \mathbb{G}$, an n-bit integer $\kappa = \sum_{i=0}^{n-1} k_i 2^i$
Output: x^κ

$R_0 \leftarrow 1_{\mathbb{G}}$; $R_1 \leftarrow x$;
$U_0 \leftarrow x^{-1}$; $U_1 \leftarrow x$;
for $i = n-1$ **down to** 0 **do**
  $R_0 \leftarrow R_0 \cdot R_1$;
  $R_1 \leftarrow R_0 \cdot U_{k_i}$;
end
return $R_{\neg k_0}$

We note that to treat k_{j+1}, $b' = k_j$. However, if we let $k_j = a_j \oplus b_j$, for $a_j, b_j \in \{0,1\}$ and $h = a_j \oplus b_j \oplus b_{j-1}$, we can modify (8) as follows:

$$(T_{0,j}, T_{1,j}) = \begin{cases} (T_{\neg h,j+1} + T_{h,j+1}, T_{0,j} - 1) & \text{if } a_j = b_j\,, \\ (T_{\neg h,j+1} + T_{h,j+1}, T_{0,j} + 1) & \text{if } a_j = \neg b_j\,. \end{cases} \tag{9}$$

By using the above equations as exponents of x, we can define Algorithm 5.

Algorithm 5: Montgomery Ladder with XOR-Split Exponent II.

Input: $x \in \mathbb{G}$, n-bit integers $A = \sum_{i=0}^{n-1} a_i 2^i$ and $B = \sum_{i=0}^{n-1} b_i 2^i$, $r \in_R \mathbb{Z}$
Output: x^κ where $\kappa = A \oplus B$

$R_0 \leftarrow 1_{\mathbb{G}}$; $R_1 \leftarrow 1_{\mathbb{G}}$; $U_0 \leftarrow x$; $U_1 \leftarrow x^{-1}$;
$b' \xleftarrow{R} \{0,1\}$; $R_{\neg b'} \leftarrow x$;
for $i = n-1$ **down to** 0 **do**
  $R_0 \leftarrow R_{b_i \oplus b'} \cdot R_{(b_i \oplus b') \oplus a_i}$;
  $R_1 \leftarrow R_0 \cdot U_{b_i}$;
  $b' \leftarrow b_i$;
end
return $R_{b'}$

Algorithm 5 follows the same sequence of instructions with the MPL. Its correctness can be verified by the fact that at every round the difference $R_0/R_1 = x$ or $R_1/R_0 = x$, as for the usual ladder step. The advantage of Algorithm 5 compared to Algorithm 2, and consequently previously proposed algorithms by Izumi et al. (Izumi et al., 2010), is the elimination of the auxiliary register R_2. Instead, the auxiliary registers U_0, U_1 manipulate the known fixed value x or x^{-1} for computational purposes, and they do not require additional computational power or updates when the algorithm is executed.

As previously, if we assume that the values held in registers $\{R_0, R_1\}$ do not leak we can state the following:

Lemma 2. *Assuming that the values held in registers $\{R_0, R_1\}$ do not leak, an implementation of Algorithm 5 is resistant to first-order side-channel analysis.*

Proof. It suffices to consider each intermediate state and verify that at least one random mask is applied. Verifying this for an entire group exponentiation would be tedious, but can be simplified if we consider two rounds of Algorithm 5. That is, if we consider round m, where $0 \le m \le n-2$, then the following operations are performed:

1. $\alpha \leftarrow b_m \oplus b'$	5. $\alpha \leftarrow b_{m+1} \oplus b_m$
2. $\beta \leftarrow \alpha \oplus a_m$	6. $\beta \leftarrow \alpha \oplus a_{m+1}$
3. $R_0 \leftarrow R_\alpha \cdot R_\beta$	7. $R_0 \leftarrow R_\alpha \cdot R_\beta$
4. $R_1 \leftarrow R_0 \cdot U_{b_m}$	8. $R_1 \leftarrow R_0 \cdot U_{b_{m+1}}$

Let the proposition $\mathcal{P}(n)$ be that round $n > 0$ is resistant to first-order side-channel analysis for the n-th treated bit of the exponent. If consider the first round, we wish to show $\mathcal{P}(1)$ is true and, in the above code fragment, b' is set to a random value from $\{0,1\}$. Then, it is easy to see that:

- the results of the operations in lines 3, 4, 7 and 8 are dependent on the random values $\{R_0, R_1\}$.
- the results of the operations in lines 1, 2, 5 and 6 are uniformly distributed on $\{0,1\}$.

If we assume that all $\mathcal{P}(m)$ is true for $m \in \{1, \ldots, n\}$, then we consider $\mathcal{P}(n+1)$ where b' is set to b_n. As b_n is one share of a previously treated exponent bit, it is indistinguishable from a random value from $\{0,1\}$. The above statements regarding the results of the operations apply. Hence, by induction we have shown $\mathcal{P}(n)$ is true for all $n > 0$. To complete the proof, we simply note that only half of the code fragment above will need to be considered in the last round. □

3.3 Boolean Scalar Splitting

In the above, we define group exponentiations applicable to any multiplicatively written group $\mathbb{G}$. However, specific groups may have particular characteristics that means the algorithms above are not suitable as described. In this section, we discuss the algorithms in the context of a group formed from the points on an elliptic curve (EC). We define the EC $\mathcal{E}$ over a finite field $\mathcal{F}_q$, for a large prime q. $\mathcal{E}$ consists of points (x,y), with x,y in $\mathcal{F}_q$, that satisfy, for example, the short Weierstraß equation

$$\mathcal{E} : y^2 = x^3 + ax + b$$

with $a,b \in \mathcal{F}_q$, and the point at infinity denoted $\boldsymbol{O}$. The set $\mathcal{E}(\mathcal{F}_q)$ is defined as $\mathcal{E}(\mathcal{F}_q) = \{(x,y) \in \mathcal{E} \,|\, x,y \in \mathcal{F}_q\} \cup \{\boldsymbol{O}\}$, where $\mathcal{E}(\mathcal{F}_q)$ forms an Abelian group under the chord-and-tangent rule and $\boldsymbol{O}$ is the identity element. Alternative equations with different representations of a neutral element are also used in cryptographic algorithms, such as Edwards curves (Edwards, 2007; Bernstein and Lange, 2009) and Montgomery curves (Montgomery, 1987). The scalar multiplication of a given point is a group exponentiation in $\mathcal{E}$ that uses elliptic curve arithmetic, i.e. addition between points or scalar multiplication $[\kappa]\boldsymbol{P}$ for some integer $\kappa < |\mathcal{E}|$, and is an important part of many cryptographic algorithms.

The algorithms presented above cannot be securely implemented as described because of the neutral element. In the short Weierstraß example, the neutral element $1_{\mathbb{G}}$ is represented in $\mathcal{E}$ as the point at infinity $\boldsymbol{O}$ and cannot be manipulated in a regular way. That is, one would typically be obliged to test for a numerical representation of $\boldsymbol{O}$ and conduct a different operation if it is detected. In practice, one would implement the algorithm such that the most significant bit (assumed to be set to one) is already treated by the pre-processing. For example, Algorithm 2 can be implemented as shown in Algorithm 6, and Algorithm 5 as shown in Algorithm 7.

As previously, if we assume that the values held in registers $\{R_0, R_1, R_2\}$ do not leak we can state the following:

Corollary 1. *Lemma 1 implies that an implementation of Algorithm 6 is resistant to first-order side-channel analysis.*

Corollary 2. *Lemma 2 implies that an implementation of Algorithm 7 is resistant to first-order side-channel analysis.*

The exponent splitting methods detailed in this paper do not modify the intermediate states generated and one would expect that randomizing projective points

Algorithm 6: Montgomery Ladder with XOR-Split Scalar on an EC.

Input: $\mathcal{E}$, $\mathcal{F}_q$, $\boldsymbol{P} \in \mathcal{E}$, n-bit integers $A = \sum_{i=0}^{n-1} a_i 2^i$ and $B = \sum_{i=0}^{n-1} b_i 2^i$
Output: $\boldsymbol{Q} = [\kappa]\boldsymbol{P}$ where $\kappa = A \oplus B$

$\boldsymbol{R}_0 \leftarrow \boldsymbol{P}$; $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$; $\boldsymbol{R}_2 \leftarrow \boldsymbol{P}$;
$b' \xleftarrow{R} \{0,1\}$;
$\boldsymbol{R}_{\neg b'} \leftarrow 2\boldsymbol{P}$;
for $i = n-2$ **down to** 0 **do**
  $\boldsymbol{R}_2 \leftarrow \boldsymbol{R}_{a_i} + \boldsymbol{R}_{\neg a_i}$;
  $\boldsymbol{R}_{a_i} \leftarrow 2\boldsymbol{R}_{(b_i \oplus b') \oplus a_i}$;
  $\boldsymbol{R}_{\neg a_i} \leftarrow \boldsymbol{R}_2$;
  $b' \leftarrow b_i$;
end
return $\boldsymbol{R}_{b'}$

Algorithm 7: Montgomery Ladder with XOR-Split Scalar II on an EC.

Input: $\mathcal{E}$, $\mathcal{F}_q$, $\boldsymbol{P} \in \mathcal{E}$, n-bit integers $A = \sum_{i=0}^{n-1} a_i 2^i$ and $B = \sum_{i=0}^{n-1} b_i 2^i$
Output: $\boldsymbol{Q} = [\kappa]\boldsymbol{P}$ where $\kappa = A \oplus B$

$\boldsymbol{R}_0 \leftarrow \boldsymbol{P}$; $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$;
$\boldsymbol{U}_0 \leftarrow \boldsymbol{P}$; $\boldsymbol{U}_1 \leftarrow -\boldsymbol{P}$;
$b' \xleftarrow{R} \{0,1\}$;
$\boldsymbol{R}_{\neg b'} \leftarrow 2\boldsymbol{P}$;
for $i = n-2$ **down to** 0 **do**
  $\boldsymbol{R}_0 \leftarrow \boldsymbol{R}_{b_i \oplus b'} + \boldsymbol{R}_{(b_i \oplus b') \oplus a_i}$;
  $\boldsymbol{R}_1 \leftarrow \boldsymbol{R}_0 + \boldsymbol{U}_{b_i}$;
  $b' \leftarrow b_i$;
end
return $\boldsymbol{R}_{b'}$

would be adequate to provide a secure solution (Win et al., 1998). However, such multiplicative masking can be problematic if an attacker can choose and input that could produce a point with a coordinate set to zero, which cannot be blinded using a multiplication (Goubin, 2003). Hence, one would need to combine our algorithms with Coron's countermeasures (Coron, 1999) and add a small multiple of the order of the group to the private key before it is used. The bit length of the multiplier needs to be chosen such that an attacker cannot predict the location of a zero-coordinate with sufficient reliability to make it visible in a side-channel attack. Having a 16-bit multiplier may be sufficient, depending on the signal-to-noise ratio of the platform. The advantage of combining these countermeasures is that one does not need to consider the longest runs of ones or zeros in the order of the group.

4 SECURITY EVALUATION

In this section, we discuss the security of the algorithms presented previously, first by making a comparison with the state-of-the-art algorithms and then by providing a security evaluation of Algorithm 2, proposed in this paper.

4.1 State-of-the-Art Comparison

In this section, we compare our proposed algorithms with a selection of algorithms discussed in the previous sections and summarize our observations in Table 1.

The first block of algorithms in Table 1, contain exponentiation algorithms using the Montgomery power ladder without splitting the exponent (Algorithm 1), with additive splitting or with variations of XOR-splitting (Algorithms 2, 3, 4). Multiplicative or Euclidean splitting are not included in this table, because in terms of security they have the same side-channel resistance as an algorithm with additive splitting. In terms of performance, the number of operations is the similar, unless the values $s^{k'}$ are precomputed and stored in memory.

The second block of algorithms summarizes the behavior of the corresponding scalar multiplication algorithms[2]. Algorithms 8 and 9 are presented in Section 5.

We note that none of the algorithms in their current form can prevent leakage from observing the intermediate values. However, intermediate values can be blinded with a random value as previously described.

4.2 Mutual Information-based Security Evaluation

Having established that the proposed exponent splitting algorithms are probing-secure against first-order side-channel attacks, we proceed to analyze the noise amplification stage of the proposed countermeasure. Analytically, we perform an evaluation of Boolean exponent splitting (as described by Algorithm 2) using the information-theoretic framework of Standaert et al. (Standaert et al., 2009). Analogous approaches can be conducted for all exponent splitting algorithms, yielding very similar results. Our analysis considers two sources of leakage, namely data-based leakage and location-based leakage (also known as address leakage). Using these two leakage sources, we demonstrate three possible attack paths against Algorithm 5, covering all possible combinations between leakage sources. Thus we show the noise amplification stage when only data-based leakage is exploited (data attack), when only location-based leakage is exploited (location attack) and finally the noise amplification stage when the adversary combines data and location leakage (hybrid attack).

4.2.1 Notation & MI Metric

In this subsection, random variables are denoted with capital letters. Instances of random variables and constant values are denoted with lowercase letters. Capital bold letters are used for random variable vectors and matrices and calligraphic font denotes sets. All simulations in this section are carried out with the identity leakage function. Observable data-based leakages of a certain intermediate value v are denoted using subscript L_v. Likewise, observable location-based leakages caused by accessing register R_i (where i the index) are denoted using subscript $L_{R\text{-}i}$. To distinguish between data-based leakage and location-based leakage we use superscript L^{data} and L^{loc}. In addition, we assume that different sources of leakage (data, location) have different noise levels i.e. we assume homoscedastic data noise $N^{data} \sim \mathcal{N}(0, \sigma^2_{data})$ and homoscedastic location noise $N^{loc} \sim \mathcal{N}(0, \sigma^2_{loc})$. We use the following formula to compute the MI metric.

$$MI(S;\mathbf{L}) = H[S] + \sum_{s \in \mathcal{S}} Pr[s] \cdot \sum_{\mathbf{m} \in \mathcal{M}^d} Pr[\mathbf{m}] \cdot d \tag{10}$$

where $d = \int_{\mathbf{l} \in \mathcal{L}^{(d+1)}} Pr[\mathbf{l}|s,\mathbf{m}] \cdot log_2 Pr[s|\mathbf{l}] \quad d\mathbf{l}$ and $Pr[s|\mathbf{l}] = \frac{\sum_{\mathbf{m}^* \in \mathcal{R}} Pr[\mathbf{l}|s,\mathbf{m}^*]}{\sum_{s^* \in \mathcal{S}} \sum_{\mathbf{m}^* \in \mathcal{R}} Pr[\mathbf{l}|s^*,\mathbf{m}^*]}$, and random variable S denotes the secret exponent bit, $\mathbf{L}$ denotes the leakage vector and $\mathbf{M}$ is a d-dimensional randomness vector that we need to sum over when randomization is in place, i.e. d is the attack order.

4.2.2 Data Leakage Attack

The first obvious way to recover k_{n-1} is by observing the data leakage of the values b_{n-1} and a_{n-1} at the same time. We run the algorithm for the first two rounds and note the intermediate values that can leak information. We let b' be a random value from R$\{0,1\}$, then:

[2] We do not count XORs, which can be implemented almost for "free" compared to the cost of multiplications (M), squaring operations (S) and modular inversions (I) in the chosen field or point additions (A) and doubling operations (D) on an elliptic curve. The subtraction of points on an elliptic curve has the same cost as an addition, so we do not count them separately.

Table 1: Comparison Table.

Algorithm	♯operations	♯registers	Hide length	ADPA	Interm. Values
Algorithm 1	$n \cdot M + n \cdot S$	2	✗	✗	✗
Clavier-Joye (Clavier and Joye, 2001)	$2(n \cdot M + n \cdot S)$	2	✗	✗	✗
Algorithm 2	$n \cdot M + n \cdot S$	3	✓	✓	✗
Algorithms 3–4	$2n \cdot M$	4	✓	✓	✗
Algorithm 5	$2n \cdot M$	4	✓	✓	✗
Algorithm 6	$(n-1) \cdot A + (n-1) \cdot D$	3	✓	✓	✗
Algorithm 7	$2 \cdot (n-1) \cdot A$	4	✓	✓	✗
Itoh et al. (Itoh et al., 2003) Alg. 8	$(n-1) \cdot D + (n-1) \cdot A + 1 \cdot I$	3	✗	✓	✗
Izumi et al. (Izumi et al., 2010) Alg. 2	$(n-1) \cdot D + (n-1) \cdot A$	3	✗	✓	✗
Algorithm 8	$n \cdot D + n \cdot A$	3	✓	✓	✗
Algorithm 9	$2 \cdot n \cdot A$	4	✓	✓	✗

$i = n-1$	$i = n-2$
1. $b_m = b_{n-1} \oplus b'$	6. $b_m = b_{n-2} \oplus b'$
2. $a_m = b_m \oplus a_{n-1}$	7. $a_m = b_m \oplus a_{n-2}$
3. $R_0 = R_{b_m} \cdot R_{a_m}$	8. $R_0 = R_{b_m} \cdot R_{a_m}$
4. $R_1 = R_0 \cdot U_{b_{n-1}}$	9. $R_1 = R_0 \cdot U_{b_{n-2}}$
5. $b' = b_{n-1}$	10. $b' = b_{n-2}$

As can be observed in above, the value b_{n-1} is accessed in the first iteration ($i = n-1$) three times, once when b_m is calculated (line 1), once implicitly for the index of $U_{b_{n-1}}$ (line 4) and finally for b' (line 5). The value a_{n-1} is accessed once during the first iteration ($i = n-1$) and it is not used in the second iteration ($i = n-2$). We notice that the value b_{n-1} is used implicitly again in the second iteration, since it is equal to b'. An attacker observing the power leakage of this algorithm should be able to probe at two different points in time, in order to observe both leakages $L^{data}_{a_{n-1}}$, $L^{data}_{b_{n-1}}$ and eventually the key, i.e. we conclude that a second-order attack is possible for this scheme. Note also that the an adversary with ability to conduct horizontal side-channel attacks (Battistello et al., 2016) could observe the leakage of b_{n-1} multiple times, average them by computing $\bar{L}^{data}_{b_{n-1}} = \frac{1}{4} * \sum_{j=1}^{4} L^{data}_{b_{n-1}}$ in order to reduce the noise level and finally perform a second-order attack. The results of the MI evaluation are visible in Figure 1. As expected, the exponent splitting scheme performs noise amplification and has a different slope compared to an unprotected exponentiation (Algorithm 1). In addition, we observe the curve's horizontal shift to the right caused by the horizontal exploitation of the available leakage, i.e. we can quantify the effect of multiple leaky points for b_{n-1}.

4.2.3 Location Leakage Attack

Let us assume that the adversary can distinguish between the manipulation of registers according to which address is accessed, similar to the address-bit DPA attack described in (Izumi et al., 2010). If the adversary can distinguish between accesses to, U_0 and U_1 for example, a direct consequence is recovery of value b_{n-1}. To mount a successful attack against Algorithm 5 using solely location-based leakage, we need the simultaneous observation of the address of U_{i_1} and R_{i_2} and R_{i_3}, for indexes $i_1 = b_{n-1}$ (line 4) and $i_2 = b_m$ (line 3) and $i_3 = a_m$ (line 3). Thus, in order to recover k_{n-1}, we need to observe leakage vector $L^{loc} = [L^{loc}_{U\text{-}i_1}, L^{loc}_{R\text{-}i_2}, L^{loc}_{R\text{-}i_3}]$, i.e. perform a third-order attack. The results are visible in Figure 2, where we can observe the noise amplification effect that increases the curve's slope. Naturally, a third-order attack using only location-based leakage tends to be less effective compared to a second-order attack using only data-based leakage. However, depending on the device, exploiting the address dependency may be more effective than exploiting the data dependency. That is, the third-order attack can become more efficient if $\sigma_{data} > \sigma_{loc}$.

4.2.4 Hybrid Leakage Attack

Lastly, we analyze the scenario in which an adversary can observe both data-based and location-based leakage. Using this information the adversary can use leakage vector

$$\mathbf{L} = [L^{data}_{a_{n-1}}, L^{loc}_{U\text{-}b_{n-1}}]$$

to carry out a second-order attack that uses data leakage to recover bit a_{n-1} and location leakage with regard to register U to recover bit b_{n-1}. Since data and location leakage imply different noise levels, i.e. ($\sigma_{data} \neq \sigma_{loc}$), we need to represent the available information as a three-dimensional plot, as in Figure 3. The wave-like plot quantifies the attainable information with regard to a particular data and location noise level. Thus, it assists the side-channel evaluator to analyze the scheme's security in a more holistic way that factors in location leakage and demonstrates the

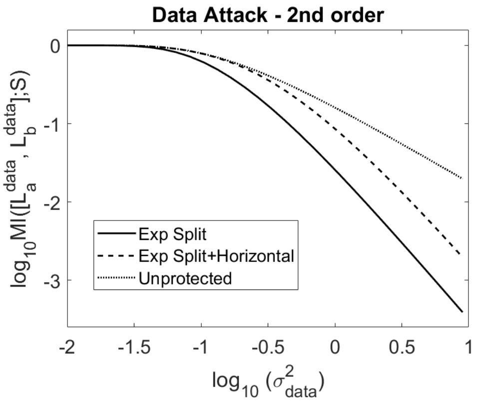

Figure 1: MI evaluation for Algorithm 2, using a data leakage attack, with and without horizontal exploitation. Observed leakage vector $\mathbf{L} = [L^{data}_{a_{n-1}}, L^{data}_{b_{n-1}}]$.

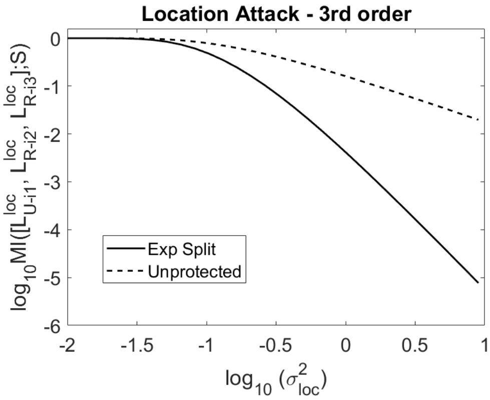

Figure 2: MI evaluation for Algorithm 2, using a location leakage attack. Observed leakage vector $\mathbf{L} = [L^{loc}_{U\text{-}i_1}, L^{loc}_{R\text{-}i_2}, L^{loc}_{R\text{-}i_3}]$.

tradeoff between data noise and location noise. If for instance $\sigma_{loc} \ll \sigma_{data}$ in the target device, the adversary can directly opt for the hybrid attack, instead of pursing a data-only attack route.

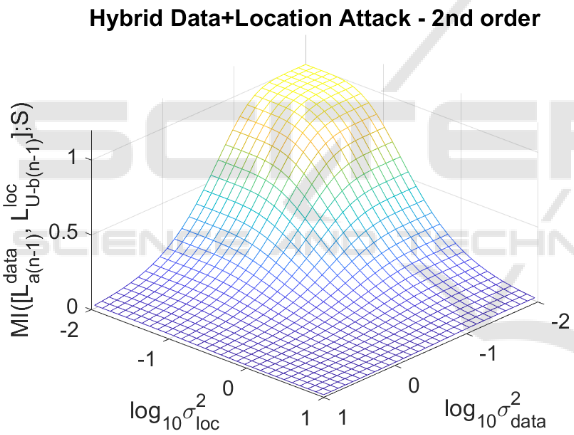

Figure 3: MI evaluation for Algorithm 5 exponent splitting, using a hybrid leakage attack. Observed leakage vector $\mathbf{L} = [L^{data}_{a_{n-1}}, L^{loc}_{U\text{-}b_{n-1}}]$.

5 IMPLEMENTATION CONSIDERATIONS

In this section, we describe the results of applying Test Vector Leakage Assessment (TVLA) (Goodwill et al., 2011) to implementations of some of the algorithms above. We further describe modifications required to achieve a secure implementation where the hardware architecture can mean that variables that should be independent leak at the same time, potentially unmasking a secret value (Balasch et al., 2015).

Our implementations were developed using Xilinx's Zynq zc702 evaluation board. The Zynq zc702 microprocessor contains two ARM7 cores and an FPGA fabric. We used one ARM7 core for our implementations, clocked at 667 MHz, and the FPGA provided a means of triggering an oscilloscope at a convenient point in our implementations. We acquired a trace of the electromagnetic emanations around one of the coupling capacitors.

The test that we used from TVLA is to determine whether there are statistically significant differences in the mean traces of two sets of traces, one acquired with a fixed scalar and the other with random scalar. One would typically randomly interleave acquisitions so that environmental effects are the same for both sets and there are no erroneous indications of leakage, caused, for example, by the least significant bit of a variable used to count the number of acquisitions. In applying this, one would take two sets of data, and conduct Welch's t-test point-by-point to determine whether there is evidence against the null hypothesis that the sets are the same. We determine that leakage is present if we observe values above 6.63σ which gives the probability of indicating leakage where no leakage is present, often referred to as a Type I error, of approximately 1×10^5 when using traces containing 3×10^5 samples. The interested reader is referred to Goodwill et al. (Goodwill et al., 2011) and Schneider and Moradi (Schneider and Moradi, 2015) for a thorough description.

We made a straightforward implementation of Algorithm 6 using NIST's P192 curve and conducted a test where we compared a set of traces with a fixed scalar compared to a set of traces with a random scalar. The elliptic curve points were implemented as homogeneous projective points. We use the x and z-coordinates in conjunction with so-called x-only algorithms for point arithmetic (Brier and Joye, 2002), as one would for an implementation of ECDH. The instantaneous electromagnetic emanations around the targeted capacitor were measured during the execution of the first 20 rounds of the implementation. The top-left trace in Figure 4 shows the result of a TVLA analysis with 1×10^3 traces where leakage can be seen in numerous places.

A straightforward implementation of Algorithm 6 was tested in the same way. The algorithm is similar to that proposed by Izumi et al. (Itoh et al., 2003) but with masking conducted before the execution of the scalar multiplication, rather than on-the-fly. The resulting TVLA traces is shown in the top-right of Figure 4, where we note that significant leakage is present with 1×10^6 traces. This is caused by the microprocessor combining values held in registers because of the architecture chosen by the designers (Balasch et al., 2015).

A more secure implementation can be made by computing some of the required indices before the execution of the main loop of the scalar multiplication, as shown in Algorithm 8. We set C to $B \oplus \left\lfloor \frac{B}{2} \right\rfloor$ such that individual bits of B are masked by adjacent bits. The resulting TVLA trace is shown in the bottom-left of Figure 4, where we observe that there is only one place where we see significant leakage with 1×10^6 traces. This leakage occurs because the initial state of $\{\boldsymbol{R}_0, \boldsymbol{R}_1\}$ contain $\{\boldsymbol{P}, 2\boldsymbol{P}\}$ in some random order. In the first loop of the scalar multiplication $\{\boldsymbol{R}_0, \boldsymbol{R}_1\}$ is overwritten with $\{2\boldsymbol{P}, 3\boldsymbol{P}\}$ or $\{3\boldsymbol{P}, 4\boldsymbol{P}\}$, in some random order, depending on whether the second most-significant bit of κ is set to 0 or 1, respectively. When $2\boldsymbol{P}$ overwrites $2\boldsymbol{P}$ the side-channel leakage will be significantly different to any other possible combination, since the Hamming distance will be zero.

A fully secure implementation can be achieved by randomizing the point produced by the doubling operation, by multiplying the x and z-coordinate of the resulting point by a random value. In implementing Algorithm 8, this was achieved by randomizing $\boldsymbol{R}_0$ and $\boldsymbol{R}_1$ before the main loop of the scalar multiplication. The resulting TVLA trace is shown in the bottom-right of Figure 4, where we observe that there is no significant leakage with 1×10^6 traces. An alternative would be to set the coordinates of $\boldsymbol{R}_{a_i}$ to zero before setting $\boldsymbol{R}_{a_i}$ to $\boldsymbol{R}_2$. Algorithm 9 shows the same arguments applied to Algorithm 7. However there is no need to randomize any points during the loops of scalar multiplication. If the redundant representation of the point assigned to $\boldsymbol{R}_0$ and $\boldsymbol{R}_1$ is randomized separately to that applied to $\boldsymbol{U}_0$ an overwrite with a Hamming distance of zero cannot occur.

Algorithm 8: Montgomery Ladder with XOR-Split Scalar on EC.

Input: $\mathcal{E}$, $\mathcal{F}_q$, $\boldsymbol{P} \in \mathcal{E}$, n-bit integers $A = \sum_{i=0}^{n-1} a_i 2^i$, $B = \sum_{i=0}^{n-1} b_i 2^i$
Output: $\boldsymbol{Q} = [\kappa]\boldsymbol{P}$ where $\kappa = A \oplus B$
Uses: $C = \sum_{i=0}^{n-1} c_i 2^i$

$\boldsymbol{R}_0 \leftarrow \boldsymbol{P}$; $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$; $\boldsymbol{R}_2 \leftarrow \boldsymbol{P}$;
$C \leftarrow B \oplus \left\lfloor \frac{B}{2} \right\rfloor$;
$b' \leftarrow b_{n-1}$;
$\boldsymbol{R}_{\neg b'} \leftarrow 2\boldsymbol{P}$;
for $i = n-2$ **down to** 0 **do**
  $\boldsymbol{R}_2 \leftarrow \boldsymbol{R}_{a_i} + \boldsymbol{R}_{\neg a_i}$;
  $\boldsymbol{R}_{a_i} \leftarrow 2\boldsymbol{R}_{a_i \oplus c_i}$;
  $\boldsymbol{R}_{\neg a_i} \leftarrow \boldsymbol{R}_2$;
end
return $\boldsymbol{R}_{b_0}$

Algorithm 9: Montg. Ladder with XOR-Split Scalar II on EC.

Input: $\mathcal{E}$, $\mathcal{F}_q$, $\boldsymbol{P} \in \mathcal{E}$, n-bit integers $A = \sum_{i=0}^{n-1} a_i 2^i$, $B = \sum_{i=0}^{n-1} b_i 2^i$
Output: $\boldsymbol{Q} = [\kappa]\boldsymbol{P}$ where $\kappa = A \oplus B$
Uses: $C = \sum_{i=0}^{n-1} c_i 2^i$ and $D = \sum_{i=0}^{n-1} d_i 2^i$

$\boldsymbol{R}_0 \leftarrow \boldsymbol{P}$; $\boldsymbol{R}_1 \leftarrow \boldsymbol{P}$;
$\boldsymbol{U}_0 \leftarrow \boldsymbol{P}$; $\boldsymbol{U}_1 \leftarrow -\boldsymbol{P}$;
$C \leftarrow B \oplus \left\lfloor \frac{B}{2} \right\rfloor$; $D \leftarrow C \oplus A$;
$b' \leftarrow b_{n-1}, \boldsymbol{R}_{\neg b'} \leftarrow 2\boldsymbol{P}$;
for $i = n-2$ **down to** 0 **do**
  $\boldsymbol{R}_0 \leftarrow \boldsymbol{R}_{c_i} + \boldsymbol{R}_{d_i}$;
  $\boldsymbol{R}_1 \leftarrow \boldsymbol{R}_0 + \boldsymbol{U}_{b_i}$;
  $b' \leftarrow b_i$;
end
return $\boldsymbol{R}_{b_0}$

6 CONCLUSIONS

In this paper, we show how an exponent can be split into two shares, where the exponent is the XOR sum of the two shares and the cost is typically an extra register and some register copies per bit. A significant advantage over previously proposed exponent splitting methods, which can have a prohibitive impact on performance (Clavier and Joye, 2001). Our method can also be applied to groups whose order contains long runs of bits set to 0 or 1 without any penalty on performance or security. Indeed, one does not need to know the order of the group; a significant advantage if, for example, one wished to implement RSA

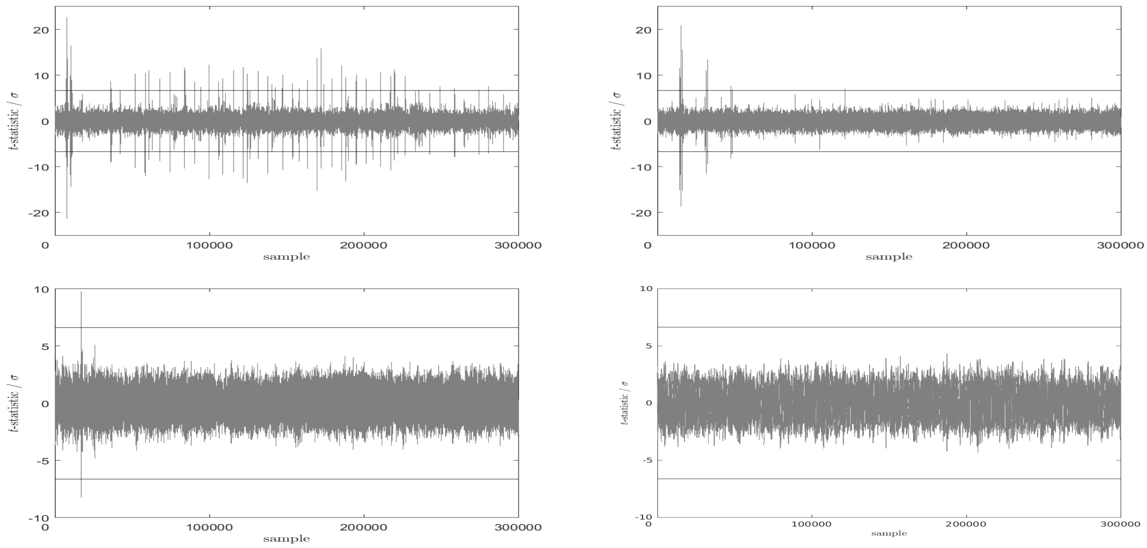

Figure 4: From top left to bottom right we show: an unmasked implementation showing leakage after 1×10^3 traces, a naïve implementation of Algorithm 6 and a more secure variant both showing leakage after 1×10^6 traces, and an implementation of Algorithm 8 that does not show leakage after 1×10^6 traces.

without using the Chinese remainder theorem.

We show that our algorithms are secure using formal methods, MI-based evaluation and TVLA on an implementation of Boolean exponent splitting. We note that our method does not prevent an attacker from using the intermediate states generated by the algorithms as a means of attack. However, inexpensive solutions such as randomizing projective points (Win et al., 1998) or Ebeid and Lambert's blinding method for RSA (Ebeid and Lambert, 2010) can be combined with our method to provide a high level of side-channel resistance.

The algorithms presented above will be more efficient than adding a multiple of the group order to the exponent, since the bit length of the exponent is not increased. Moreover, the resistance to collision attacks is superior, since one would need to conduct several attacks to derive each share and reconstruct the exponent. Where one is adding a random multiple of the exponent any bits recovered directly relate to bits of the exponent used. It has not yet been shown that one can derive an exponent from gaining partial information on a series of blinded exponents, but significant advances have been made (Schindler and Itoh, 2011; Joye and Lepoint, 2012; Schindler, 2014; Schindler and Wiemers, 2014).

ACKNOWLEDGEMENTS

The authors would like to thank Lauren De Meyer and Michael Hamburg for their helpful comments.

REFERENCES

Balasch, J., Gierlichs, B., Grosso, V., Reparaz, O., and Standaert, F. (2015). On the cost of lazy engineering for masked software implementations. In *CARDIS 2014*, volume 8968 of *LNCS*, pages 64–81. Springer.

Battistello, A., Coron, J., Prouff, E., and Zeitoun, R. (2016). Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In *CHES 2016*, volume 9813 of *LNCS*, pages 23–39. Springer.

Bernstein, D. J. and Lange, T. (2009). A complete set of addition laws
for incomplete edwards curves. Cryptology ePrint Archive, Report 2009/580. https://eprint.iacr.org/2009/580.

Brier, E., Clavier, C., and Olivier, F. (2004). Correlation power analysis with a leakage model. In *CHES 2004*, volume 3156 of *LNCS*, pages 16–29. Springer.

Brier, E. and Joye, M. (2002). Weierstraß elliptic curves and side-channel attacks. In *PKC 2002*, volume 2274 of *LNCS*, pages 335–345. Springer.

Chari, S., Jutla, C. S., Rao, J. R., and Rohatgi, P. (1999). Towards sound approaches to counteract power-analysis attacks. In *CRYPTO '99*, volume 1666 of *LNCS*, pages 398–412. Springer.

Ciet, M. and Joye, M. (2003). (virtually) free randomization techniques for elliptic curve cryptography. In *ICICS 2003*, volume 2836 of *LNCS*, pages 348–359. Springer.

Clavier, C. and Joye, M. (2001). Universal exponentiation algorithm. In *CHES 2001*, volume 2162 of *LNCS*, pages 300–308. Springer.

Coron, J. (1999). Resistance against differential power analysis for elliptic curve cryptosystems. In *CHES 1999*, volume 1717 of *LNCS*, pages 292–302. Springer.

Ebeid, N. M. and Lambert, R. (2010). A new CRT-RSA algorithm resistant to powerful fault attacks. In *WESS 2010*, page 8. ACM.

Edwards, H. M. (2007). A normal form for elliptic curves. In *Bulletin of the American Mathematical Society*, volume 44, pages 393–422.

Feix, B., Roussellet, M., and Venelli, A. (2014). Side-channel analysis on blinded regular scalar multiplications. In *INDOCRYPT 2014*, volume 8885 of *LNCS*, pages 3–20. Springer.

Gandolfi, K., Mourtel, C., and Olivier, F. (2001). Electromagnetic analysis: Concrete results. In *CHES 2001*, volume 2162 of *LNCS*, pages 251–261. Springer.

Goodwill, G., Jun, B., Jaffe, J., and Rohatgi, P. (2011). A testing methodology for side channel resistance validation. NIST non-invasive attack testing workshop.

Goubin, L. (2003). A refined power-analysis attack on elliptic curve cryptosystems. In *PKC 2003*, volume 2567 of *LNCS*, pages 199–210. Springer.

Hanley, N., Kim, H., and Tunstall, M. (2015). Exploiting collisions in addition chain-based exponentiation algorithms using a single trace. In *CT-RSA 2015*, volume 9048 of *LNCS*, pages 431–448. Springer.

Itoh, K., Izu, T., and Takenada, M. (2002). Address-bit differential power analysis of cryptographic schemes OK-ECDH and OK-ECDSA. In *CHES 2002*, volume 2523 of *LNCS*, pages 129–143. Springer.

Itoh, K., Izu, T., and Takenada, M. (2003). A practical countermeasure against address-bit differential power analysis. In *CHES 2003*, volume 2779 of *LNCS*, pages 382–396. Springer.

Izumi, M., Ikegami, J., Sakiyama, K., and Ohta, K. (2010). Improved countermeasures against address-bit DPA for ECC scalar multiplication. In *DATE 2010*, pages 981–984. IEEE.

Joye, M. and Lepoint, T. (2012). Partial key exposure on RSA with private exponents larger than N. In *ISPEC 2012*, volume 7232 of *LNCS*, pages 369–380. Springer.

Joye, M. and Villegas, K. (2002). A protected division algorithm. In *CARDIS 2002*. USENIX.

Joye, M. and Yen, S.-M. (2002). The Montgomery powering ladder. In *CHES 2002*, volume 2523 of *LNCS*, pages 291–302. Springer.

Kim, H., Kim, T. H., Yoon, J. C., and Hong, S. (2010). Practical second-order correlation power analysis on the message blinding method and its novel countermeasure for RSA. *ETRI Journal*, 32(1):102–111.

Kocher, P. (1996). Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *CRYPTO '96*, volume 1109 of *LNCS*, pages 104–113. Springer.

Kocher, P., Jaffe, J., and Jun, B. (1999). Differential power analysis. In *CRYPTO '99*, volume 1666 of *LNCS*, pages 388–397. Springer.

Le, D.-P., Tan, C.-H., and Tunstall, M. (2015). Randomizing the Montgomery powering ladder. In *WISTP 2015*, volume 9311 of *LNCS*, pages 155–170. Springer.

Messerges, T. S. and Dabbish, E. A. (1999). Investigations of power analysis attacks on smartcards. In *Smartcard 1999*. USENIX Association.

Messerges, T. S., Dabbish, E. A., and Sloan, R. H. (1999). Power analysis attacks of modular exponentiation in smartcards. In *CHES'99*, volume 1717 of *LNCS*, pages 144–157. Springer.

Montgomery, P. L. (1987). Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48(177):243–264.

National Institute of Standards and Technology (NIST) (2009). Recommended elliptic curves for federal government use. In the appendix of FIPS 186-3, available from http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf.

Quisquater, J.-J. and Samyde, D. (2001). Electromagnetic analysis (EMA): Measures and counter-measures for smart cards. In *E-smart 2001*, volume 2140 of *LNCS*, pages 200–210. Springer.

Rivest, R., Shamir, A., and Adleman, L. M. (1978). Method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126.

Schindler, W. (2014). Exclusive exponent blinding may not suffice to prevent timing attacks on RSA. Cryptology ePrint Archive, Report 2014/869. http://eprint.iacr.org/.

Schindler, W. and Itoh, K. (2011). Exponent blinding does not always lift (partial) SPA resistance to higher-level security. In *ACNS 2011*, volume 6715 of *LNCS*, pages 73–90. Springer.

Schindler, W. and Wiemers, A. (2014). Power attacks in the presence of exponent blinding. *J. Cryptographic Engineering*, 4(4):213–236.

Schneider, T. and Moradi, A. (2015). Leakage assessment methodology - A clear roadmap for side-channel evaluations. In *CHES 2015*, volume 9293 of *LNCS*, pages 495–513. Springer.

Smart, N., Oswald, E., and Page, D. (2008). Randomised representations. *IET Proceedings on Information Security*, 2(2):19–27.

Standaert, F., Malkin, T., and Yung, M. (2009). A unified framework for the analysis of side-channel key recovery attacks. In *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 443–461. Springer.

Win, E. D., Mister, S., Preneel, B., and Wiener, M. J. (1998). On the performance of signature schemes based on elliptic curves. In *ANTS 1998*, volume 1423 of *LNCS*, pages 252–266. Springer.

Witteman, M. F., van Woudenberg, J. G. J., and Menarini, F. (2011). Defeating RSA multiply-always and message blinding countermeasures. In *CT-RSA 2011*, volume 6558 of *LNCS*, pages 77–88. Springer.